State-based representation of CCSL operators

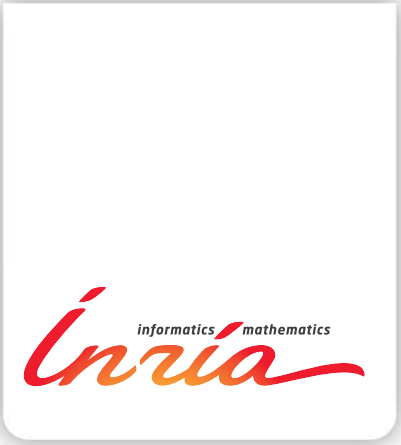
Frédéric Mallet, Jean-Vivien Millo, Yuliia Romenska

State-based representation of CCSL operators

Frédéric Mallet, Jean-Vivien Millo, Yuliia Romenska

RESEARCH REPORT
N° 8334
July 2013
Project-Team Aoste

ISSN 0249-6399 ISRN INRIA/RR--8334--FR+ENG



State-based representation of CCSL operators

Frédéric Mallet*, Jean-Vivien Millo†, Yuliia Romenska

Project-Team Aoste

Research Report n° 8334 — July 2013 — 26 pages

Abstract: The UML Profile for Modeling and Analysis of Real-Time and Embedded systems promises a general modeling framework to design and analyze systems. Lots of works have been published on the modeling capabilities offered by MARTE, much less on verification techniques supported. The Clock Constraint Specification Language (CCSL), first introduced as a companion language for MARTE, was devised to offer a formal support to conduct causal and temporal analyses on MARTE models.
This work introduces formally a state-based semantics for CCSL operators.

Key-words: Logical Time, UML MARTE, CCSL, infinite transition systems

* Université Nice Sophia Antipolis
† This work has been partially funded by ARTEMIS Grant $N°269362$ – Project PRESTO - http://www.presto-embedded.eu

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Les opérateurs CCSL sous forme de systèmes de transitions infinis

Résumé : Le profil UML pour la modélisation et l'analyse de syst'emes temps réel et embarqués (MARTE) promet d'être un environnement général pour la conception et l'analyse de syst'emes. De nombreux travaux ont présenté les capacités de modélisation de MARTE, beaucoup moins ont présenté les capacités de vérification exhaustive. Le langage CCSL (Clock Constraint Specification Language) a été initialement introduit comme une annexe de MARTE pour offrir un support à la vérification formelle de propriétés causales et temporelles sur des modèles MARTE.

Ce travail introduit formellement une sémantique à base de systèmes de transitions étiquetées. C'est une étape importante pour permettre l'analyse exhaustive de modèles MARTE/CCSL.

Mots-clés : temps-logique, UML MARTE, CCSL, systèmes de transitions infinis

1 Introduction

The UML Profile for Modeling and Analysis of Real-Time and Embedded systems [1] (MARTE), adopted in November 2009, has introduced a *Time model* [2] that extends the informal *Simple Time* of The Unified Modeling Language (UML 2.x). This time model is general enough to support different forms of time (discrete or dense, chronometric or logical). Its so-called *clocks* allow enforcing as well as observing the occurrences of events and the behavior of annotated UML elements. The time model comes with a companion language called the *Clock Constraint Specification Language* (CCSL) [3] and defined in an annex of the MARTE specification. Initially devised as a simple language for expressing constraints between clocks of a MARTE model, CCSL has evolved and has been developed independently of the UML. CCSL is now equipped with a formal semantics [3] and is supported by a software environment (TimeSquare [4][1]) that allows for the specification, solving, and visualization of clock constraints.

MARTE promises a general modeling framework to design and analyze systems. Lots of works have been published on the modeling capabilities offered by MARTE, much less on verification techniques supported. While the initial semantics of CCSL is described as a set of rewriting rules [3], this paper proposes as a first contribution a state-based semantics for each of the kernel CCSL operators. The global semantics emerging of the parallel composition of CCSL constraints then becomes the synchronized product of the automaton of each individual constraint. Since automaton for some CCSL operators can be infinite, this requires specific attention to compute the synchronized product. The second contribution is an algorithm that builds the synchronized product. The algorithm terminates when the set of states reachable through the synchronized product is finite. The third contribution is a discussion on a sufficient condition to guarantee that the synchronized product is actually finite.

Section 4 proposes a state-based semantics for CCSL. Section 5 discusses boundness issues on CCSL specifications. Section 6 illustrates the use of CCSL for architecture-driven analysis. It shows how abstract representations of the application and the architecture are built and how the two models are mapped through an allocation process. Section 7 makes a comparison with related works.

2 The Clock Constraint Specification Language

The Clock Constraint Specification Language (CCSL) has been developed to elaborate and reason on the logical time model [2] of MARTE. A technical report [3] describes the syntax and the semantics of a kernel set of CCSL constraints.

The notion of multiform logical time has first been used in the theory of Synchronous languages [5] and its polychronous extensions [6]. The use of tagged systems to capture and compare models of computations was advocated by [7]. CCSL provides a concrete syntax to make the polychronous clocks become first-class citizens of UML-like models.

A *clock* c is a totally ordered set of *instants*, $\mathcal{I}_c$. In the following, i and j are instants. A *time structure* is a set of clocks $\mathcal{C}$ and a set of relations on instants $\mathcal{I} = \bigcup_{c\in\mathcal{C}} \mathcal{I}_c$. CCSL considers two kinds of relations: *causal* and *temporal* ones. The basic causal relation is causality/*dependency*, a binary relation on $\mathcal{I}$: $\preccurlyeq \subset \mathcal{I} \times \mathcal{I}$. $i \preccurlyeq j$ means i causes j or j depends on i. $\preccurlyeq$ is a pre-order on $\mathcal{I}$, *i.e.,* it is reflexive and transitive. The basic temporal relations are *precedence* ($\prec$), *coincidence* ($\equiv$), and *exclusion* ($\#$), three binary relations on $\mathcal{I}$. For any pair of instants $(i, j) \in \mathcal{I} \times \mathcal{I}$ in a time structure, $i \prec j$ means that the only acceptable execution traces are those where i occurs strictly before j (i precedes j). $\prec$ is transitive and asymmetric (reflexive and antisymmetric).

[1] `http://timesquare.inria.fr`

$i \equiv j$ imposes instants i and j to be coincident, *i.e.,* they must occur at the same execution step, both of them or none of them. $\equiv$ is an equivalence relation, *i.e.,* it is reflexive, symmetric and transitive. $i \# j$ forbids the coincidence of the two instants, *i.e.,* they cannot occur at the same execution step. $\#$ is irreflexive and symmetric. A consistency rule is enforced between causal and temporal relations. $i \preccurlyeq j$ can be refined either as $i \prec j$ or $i \equiv j$, but j can never precede i.

In this paper, we consider discrete sets of instants only, so that the instants of a clock can be indexed by natural numbers. For a clock $c \in \mathcal{C}$, and for any $k \in \mathbb{N}_{>0}$, $c[k]$ denotes the k^{th} instant of c.

3 Definitions

3.1 Logical time model

Clocks in CCSL are used to measure dates of occurrences of events in a system. Logical clocks replace physical dates by a logical sequencing. We never presume that clocks or events are described relative to a global physical time but we rather consider that clocks are independent of each other.

Definition 1 (Logical clock) *A* clock *c belongs to a set of propositions $\mathcal{C}$.*

Clocks are assumed to be independent of each other. During the execution of a system, clocks tick according to occurrences of related events. The schedule captures what happens during one particular execution.

Definition 2 (Schedule) *A* schedule *is defined as a function $Sched : \mathbb{N}_{>0} \to 2^{\mathcal{C}}$. Given an* execution step *$s \in \mathbb{N}_{>0}$, and a schedule $\sigma \in Sched$, $\sigma(s)$ denotes the set of clocks that* tick *at step s.*

For a given schedule, it is useful to know the relative advance of clocks, *i.e.,* their configuration.

Definition 3 (Clock configuration) *For a given schedule σ, the* configuration *is defined as $\chi_\sigma : \mathcal{C} \times \mathbb{N} \to \mathbb{N}$. $\forall c \in \mathcal{C}$, it is defined recursively as:*

- $\chi_\sigma(c, 0) = 0$, *the initial configuration,*
- $\forall n > 0, \chi_\sigma(c, n) = \chi_\sigma(c, n-1)$ *if* $c \notin \sigma(n)$,
- $\forall n > 0, \chi_\sigma(c, n) = \chi_\sigma(c, n-1) + 1$ *if* $c \in \sigma(n)$.

For a clock $c \in \mathcal{C}$, and a step $n \in \mathbb{N}$, $\chi_\sigma(c, n)$ denotes the number of times the clock c has ticked at step n for the given schedule σ.

The Clock Constraint Specification Language is used to specify a set of *valid schedules*. Since a CCSL specification does not assume a global time, there is usually an infinite number of schedules that satisfy a given specification. If there is no satisfying schedule, then the specification is ill-formed.

Definition 4 (CCSL specification) *A* CCSL specification *Spec is a tuple $\langle \mathcal{C}, Rel, Def \rangle$, where $\mathcal{C}$ is a set of clocks, Rel and Def are two disjoint sets collectively called* CCSL constraints*, Rel is a set of* clock relations *whereas Def is a set of* clock definitions.

3.1.1 Clock relations

Definition 5 (Primitive CCSL relations) *We define the set of primitive relation operators:* $RelOp = \{\boxed{\subset}, \boxed{\#}, \boxed{\prec}, \boxed{\preccurlyeq}\}$.
A Clock relation *is* $Rel : \mathcal{C} \times RelOp \times \mathcal{C}$. *Let* $left : Rel \to \mathcal{C}$ *be the function that gives the left clock involved in a relation. Let* $right : Rel \to \mathcal{C}$ *be the function that gives the right clock involved in a relation. Let* $op : Rel \to RelOp$ *be the function that gives the operator involved in a relation.*

The first two relations are synchronous. They force clocks to tick or not to tick depending on whether another clock ticks or not. Subclocking prevents a subclock c_1 from ticking when its super clock c_2 does not tick. In other words, c_1 is a subclock of c_2 for a given schedule iff c_1 only ticks when c_2 ticks. Exclusion prevents two clocks from ticking simultaneously. Synchrony forces two clocks to tick always simultaneously. Their satisfaction rules are given below.

Definition 6 (Synchronous relations) *The satisfaction rules for the synchronous constraints with regards to a given schedule* σ *are:*

$$\sigma \models_{ccsl} c_1 \boxed{\subset} c_2 \iff \forall n \in \mathbb{N}_{>0}, c_1 \in \sigma(n) \implies c_2 \in \sigma(n) \quad \textbf{(Subclocking)} \tag{1a}$$

$$\sigma \models_{ccsl} c_1 \boxed{\#} c_2 \iff \forall n \in \mathbb{N}_{>0}, c_1 \notin \sigma(n) \vee c_2 \notin \sigma(n) \quad \textbf{(Exclusion)} \tag{1b}$$

Note that by definition, Subclocking is a pre-order on $\mathcal{C}$, *i.e.,* it is reflexive and transitive.

The latter two relations are asynchronous. They forbid clocks to tick depending on what has happened on other clocks in the earlier steps. Causality requires a clock c_1 to be always in advance on another clock c_2 but allows the case where the two clocks tick synchronously. Precedence is a stronger form that forbids pure Synchrony and requires c_1 to be strictly in advance on c_2.

Definition 7 (Asynchronous relations) *The satisfaction rules for the asynchronous constraints with regards to a given schedule* σ *are:*

$$\sigma \models_{ccsl} c_1 \boxed{\preccurlyeq} c_2 \iff \forall n \in \mathbb{N}, \chi_\sigma(c_1, n) - \chi_\sigma(c_2, n) \geq 0 \quad \textbf{(Causality)} \tag{2a}$$

$$\sigma \models_{ccsl} c_1 \boxed{\prec} c_2 \iff \forall n \in \mathbb{N}, (\chi_\sigma(c_1, n) = \chi_\sigma(c_2, n)) \implies c_2 \notin \sigma(n+1) \quad \textbf{(Precedence)} \tag{2b}$$

Note: Causality is another pre-order on $\mathcal{C}$.

Proposition 1 (Precedence implies causality) *The Precedence is a stronger form of causality:* $\sigma \models_{ccsl} c1 \boxed{\prec} c2 \implies \sigma \models_{ccsl} c1 \boxed{\preccurlyeq} c2$

Proof of Proposition 1 By recursion on χ_σ.
HR(n) $= \chi_\sigma(c1, n) \geq \chi_\sigma(c2, n)$.
HR(0) is true since $\chi_\sigma(c2, 0) = \chi_\sigma(c1, 0) = 0$.
Assume HR(n-1).

- If $\chi_\sigma(c1, n-1) = \chi_\sigma(c2, n-1)$ then Eq. 2b $\implies$ $c_2 \notin \sigma(n)$ $\implies$ (Def. 3) $(\chi_\sigma(c1, n) \geq \chi_\sigma(c1, n-1) \wedge \chi_\sigma(c2, n) = \chi_\sigma(c2, n-1)) \implies$ HR(n)

- If $\chi_\sigma(c1, n-1) > \chi_\sigma(c2, n-1)$. The worst case is if $c2 \in \sigma(n) \wedge c1 \notin \sigma(n)$, which implies $(\chi_\sigma(c1, n) = \chi_\sigma(c1, n-1) \wedge \chi_\sigma(c2, n) = \chi_\sigma(c2, n-1) + 1)$ and then HR(n).

3.1.2 Clock definitions

A clock definition is of the form $c \triangleq e$ where $c \in \mathcal{C}$ and e is a *clock expression*. We consider two kinds of expressions the binary expressions and the unary expressions.

Definition 8 (Primitive CCSL binary expressions) *The primitive binary expressions are* $BinExpr : \mathcal{C} \times ExprOp \times \mathcal{C}$, *where* $ExprOp = \{+, *, \wedge, \vee\}$.
We define $first : BinExpr \rightarrow \mathcal{C}$ *the function that gives the first clock involved in a binary expression.*
We define $second : BinExpr \rightarrow \mathcal{C}$ *the function that gives the second clock involved in a binary expression.*
We define $op : BinExpr \rightarrow ExprOp$ *the function that gives the operator involved in a binary expression.*

The first two clock expressions are based on Subclocking. Union builds the slowest super clock of two given clocks. Intersection builds the fastest clock that is a subclock of two given clocks.

Definition 9 (Union and intersection) *The satisfaction rules of Union and Intersection for a given schedule* σ *are:*

$$\sigma \models_{ccsl} u \triangleq c_1 + c_2 \iff \forall n \in \mathbb{N}_{>0}, u \in \sigma(n) \Leftrightarrow c_1 \in \sigma(n) \vee c_2 \in \sigma(n) (\textbf{Union}) \quad (3a)$$

$$\sigma \models_{ccsl} i \triangleq c_1 * c_2 \iff \forall n \in \mathbb{N}_{>0}, i \in \sigma(n) \Leftrightarrow c_1 \in \sigma(n) \wedge c_2 \in \sigma(n) (\textbf{Intersection}) \quad (3b)$$

The following clock expressions are based on Causality. Infimum builds the slowest clock that is faster than two given clocks. Supremum builds the fastest clock that is slower than two given clocks.

Definition 10 (Infimum and Supremum) *The satisfaction rules of Infimum and Supremum for a given schedule* σ *are:*

$$\sigma \models_{ccsl} inf \triangleq c_1 \wedge c_2 \iff \forall n \in \mathbb{N}, \chi_\sigma(inf, n) = max(\chi_\sigma(c_1, n), \chi_\sigma(c2, n)) (\textbf{Infimum}) \quad (4a)$$

$$\sigma \models_{ccsl} sup \triangleq c_1 \vee c_2 \iff \forall n \in \mathbb{N}, \chi_\sigma(sup, n) = min(\chi_\sigma(c_1, n), \chi_\sigma(c_2, n)) (\textbf{Supremum}) \quad (4b)$$

All the unary expressions are bounded, we only consider here one of them, the Delay: $e := c \$ d$, where $d \in \mathbb{N}$. This expression models a pure delay. It is used to produce a clock that is always a given number of ticks d late compared to its original clock. d is a positive integer.

Definition 11 (Delay) *The satisfaction rule of Delay for a given schedule* σ *and for a given natural number* $d \in \mathbb{N}$ *is:*

$$\sigma \models_{ccsl} del \triangleq c \$ d \iff \forall n \in \mathbb{N}, \chi_\sigma(del, n) = max(\chi_\sigma(c, n) - d, 0) (\textbf{Delay}) \quad (5)$$

To help the reader understand the semantics of the expressions, Figure 1 gives an example of schedule σ that satisfies several expressions. Check marks represent the steps where a given clock ticks.

3.2 Composition

Definition 12 (CCSL specification satisfaction) *A schedule* σ *satisfies a* CCSL *specification* $SPEC$, *iff it satisfies all of its constraints:* $\sigma \models_{ccsl} SPEC \Leftrightarrow (\forall rel \in Rel, \sigma \models_{ccsl} rel) \wedge (\forall def \in Def, \sigma \models_{ccsl} def)$

step	1	2	3	4	5	6	7
c_1	✓			✓			✓
c_2		✓		✓	✓	✓	
$u \triangleq c_1 \boxed{+} c_2$	✓	✓		✓	✓	✓	✓
$i \triangleq c_1 \boxed{*} c_2$				✓			
$inf \triangleq c_1 \boxed{\wedge} c_2$	✓			✓	✓	✓	
$sup \triangleq c_1 \boxed{\vee} c_2$		✓		✓			✓
$d \triangleq c_2 \$ 2$					✓	✓	

Figure 1: An example of schedule σ

Definition 13 (Bounded CCSL relations) *For a given* CCSL *specification $SPEC$, a relation $r \in Rel$ is bounded iff* $(\sigma \models_{ccsl} SPEC) \implies (\exists m \in \mathbb{N}, \forall n \in \mathbb{N}, |\chi_\sigma(left(r), n) - \chi_\sigma(right(r), n)| \leq m)$.

Note that, by definition of Causality and because of Proposition 1, we always have $op(r) \in \{\boxed{\prec}, \boxed{\preccurlyeq}\} \implies \forall n \in \mathbb{N}, \chi_\sigma(left(r), n) - \chi_\sigma(right(r), n) \geq 0$, so we do not have to worry about finding a lower bound.

Definition 14 (Bounded CCSL expressions) *For a given* CCSL *specification $SPEC$, a binary expression $e \in BinExpr$ is bounded iff* $(\sigma \models_{ccsl} SPEC) \implies (\exists m \in \mathbb{N}, \forall n \in \mathbb{N}, |\chi_\sigma(first(e), n) - \chi_\sigma(second(e), n)| \leq m))$. *Unary expressions are always bounded.*

In [8], we have shown that the behavior of a CCSL specification was captured by the synchronized product of the transition systems for each constraint. Obviously, when all the composed transition systems are finite, then the result is necessarily finite. However, the result can also be finite when some of the composed transition systems have an infinite number of states. This is because we only consider the states that are reachable. So safety amounts to having only a finite number of states in the product reachable from the initial state. This is equivalent to being able to bound the counters used in unbounded constraints.

Let us illustrate that on a simple example. Consider, for instance the following CCSL specification: $(c_1 \boxed{\prec} c_2) \wedge (c'_1 \triangleq c_1 \$ 1) \wedge (c_2 \boxed{\prec} c'_1)$. In this specification, the second constraint (Delay) is bounded, but the two others are unbounded. However, the result is still considered to be safe since there is only a finite number of reachable states in the synchronized product as shown in Figure 2. This comes from the fact that counters used in the two Precedences are bounded by the Delay of the second constraint. This particular composition pattern is frequently used and is called Alternation.

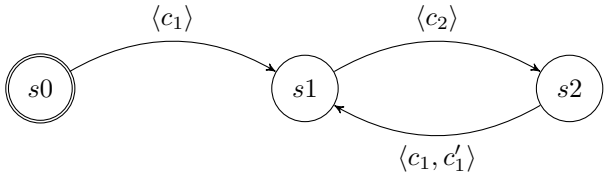


Figure 2: A safe composition of unbounded constraints

Definition 15 (Safe CCSL specification) *A* CCSL *specification is* safe *iff* $\forall \sigma, \sigma \models_{ccsl} SPEC$*:*

- *all the relations are bounded:* $\forall r \in Rel, r$ *is bounded,*
- *all the binary expressions within a clock definition are bounded:* $\forall e \in BinExpr, e$ *is bounded*

Definition 16 (Bounded precedence) *We define a new composite* CCSL *constraint called* Bounded precedence *by the following satisfaction rule* ($n \in \mathbb{N}$)*:*

$$\begin{aligned} st\sigma \models_{ccsl} c_1 \boxed{\prec_n} c_2 \iff & (\textbf{Boundedprecedence}) \\ \sigma \models_{ccsl} c_1 \boxed{\prec} & \ c_2 \\ \wedge\ \sigma \models_{ccsl} c_1' \triangleq c_1 \$ & \ n \\ \wedge\ \sigma \models_{ccsl} c_2 \boxed{\prec} & \ c_1' \end{aligned}$$

We call alternation *the case where* $n = 1$*:*

$$\sigma \models_{ccsl} c_1 \boxed{\sim} c_2 \equiv \sigma \models_{ccsl} c_1 \boxed{\prec_1} c_2 (\textbf{Alternation})$$

Proposition 2 (The bounded precedence is safe) *Let* $c = c_1 \boxed{\prec_d} c_2$*, constraint* c *is safe.*

Proof of Proposition 2 Let us take a σ such that $\sigma \models_{ccsl} c_1 \boxed{\prec_d} c_2$. The first constraint gives $\forall n \in \mathbb{N}, \chi_\sigma(c_1, n) - \chi_\sigma(c_2, n) \geq 0$. The third one gives $\forall n \in \mathbb{N}, \chi_\sigma(c_2, n) - \chi_\sigma(c_1', n) \geq 0$, so $\forall n \in \mathbb{N}, \chi_\sigma(c_1, n) - \chi_\sigma(c_1', n) \geq 0$. For the specification to be bounded, we need to show that $\exists m \in \mathbb{N}, \forall n \in \mathbb{N}, |\chi_\sigma(c_1, n) - \chi_\sigma(c_1', n)| \leq m$.
If $\chi_\sigma(c_1, n) \leq d$, then Eq. 5 gives $\chi_\sigma(c_1', n) = 0$ and therefore $\chi_\sigma(c_1, n) - \chi_\sigma(c_1', n) \leq d$.
If $\chi_\sigma(c_1, n) \geq d$, then Eq. 5 gives $\chi_\sigma(c_1', n) = \chi_\sigma(c_1, n) - d$ and also $\chi_\sigma(c_1, n) - \chi_\sigma(c_1', n) \leq d$. ∎

3.3 Safety issues

We consider an abstraction of the CCSL specification that we call a *causality clock graph*. Indeed, Causality is the foundational construct that introduces unbounded integers in a CCSL specification. Then, we use this abstraction to show that counters included in Precedence, Causality, Infimum and Supremum constraints are bounded. For that purpose, we consider the causal relations includes in a CCSL specification, but we also consider causal relations induced by other constraints. The causality clock graph captures all the causal relations, whether directly specified or induced. The remainder of this subsection discusses the induced causal relations.

Definition 17 (Causality clock graph) *A* Causality clock graph *(CCG) is a directed graph* $D = (\mathcal{C}, A, \Delta)$*.* $\mathcal{C}$ *is a set of nodes denoting clocks.* $A \subset \mathcal{C} \times \mathcal{C}$ *is a set of arcs (directed edges).* $\Delta \subset \mathcal{C} \times \mathcal{C}$ *is a set of* counter-arcs *between two clocks.*

In a CCG, an arc $a = (c_1, c_2)$ is directed from c_1 to c_2 and denotes a causality $c_1 \boxed{\preccurlyeq} c_2$. A counter-arc $\delta = (c_1, c_2)$ is used to identify a constraint that would generate an infinite number of states if left unbounded. To each counter-arc $\delta = (c_1, c_2)$, we associate a function $\delta_{c_1}^{c_2}$:

$$\begin{aligned} \delta_{c_1}^{c_2} : \mathbb{N} &\to \mathbb{N} \\ n &\mapsto \chi_\sigma(c_1, n) - \chi_\sigma(c_2, n) \end{aligned}$$

The safety analysis must show that for each counter-arc, for each schedule σ, $\exists m \in nat, \forall n \in \mathbb{N}, |\delta_{c_1}^{c_2}(n)| \leq m$.

Definition 18 (Complete causality clock graph) *Given a* CCSL *specification $SPEC$, a causality clock graph D_{SPEC} is* complete *with regards to $SPEC$ when all the causal relations implied by SPEC are captured in the graph and only those relations.* $\forall \sigma, \sigma \models_{ccsl} SPEC, \forall (c_1, c_2) \in \mathcal{C} \times \mathcal{C}$, $(\exists d \in nat, \forall n \in \mathbb{N}, \delta_{c_1}^{c_2}(n) \geq -d \Leftrightarrow (c_1, c_2)$ *is an arc in* $D_{SPEC})$

The notion of completeness is necessary to show that no causal relation has been 'forgotten' in the graph. It means that as soon as a constraint implies that the counter between two clocks can be bounded (either with a lower or an upper bound) then (and only then) there should be a counter-arc in the causality clock graph. Indeed, if arcs are missing, then the safety analysis might conclude that a graph is not safe, while a CCSL specification is actually safe.

3.4 Building the causality clock graph

Obviously, the constraint $c_1 \boxed{\preccurlyeq} c_2$ always induces a lower bound. For the CCSL specification to be bounded, we need to establish an upper bound. An arc from c_1 to c_2 denotes that we have a lower bound ($\forall n \in \mathbb{N}, \delta_{c_1}^{c_2}(n) \geq 0$). A counter-arc between c_1 and c_2 denotes that we need to establish the upper bound. More formally, for a given CCSL specification SPEC, we build the causality clock graph $D_{SPEC} = (\mathcal{C}, A, \Delta)$ such that $\forall r \in Rel, op(r) = \boxed{\preccurlyeq} \implies (left(r), right(r)) \in A \wedge (left(r), right(r)) \in \Delta$.

Building arcs only for these relations would lead to an incomplete graph. Other bounds are indeed indirectly induced by most CCSL constraints. The first obvious example is given by Proposition 1. Hence, every Precedence also leads to an arc and a counter-arc in the CCG. $\forall r \in Rel, op(r) = \boxed{\prec} \implies (left(r), right(r)) \in A \wedge (left(r), right(r)) \in \Delta$.

In the remainder of this section, the other implied causality relations are discussed.

The first family of implications comes from the relationship between Subclocking and Causality.

Proposition 3 (Subclocking implies causality) *When c_1 is a subclock of c_2 then c_2 is faster than c_1:* $\sigma \models_{ccsl} c_1 \boxed{\subset} c_2 \implies \sigma \models_{ccsl} c_2 \boxed{\preccurlyeq} c_1$

Proof of Proposition 3 By recursion on χ_σ.
HR(n) $= \chi_\sigma(c2, n) \geq \chi_\sigma(c1, n)$.
HR(0) is true since $\chi_\sigma(c2, 0) = \chi_\sigma(c1, 0) = 0$.
Assume HR(n-1).

- If $c1 \notin \sigma(n) \wedge c2 \notin \sigma(n)$ then $\chi_\sigma(c1, n) = \chi_\sigma(c1, n-1) \wedge \chi_\sigma(c2, n) = \chi_\sigma(c2, n-1)$ then HR(n).
- If $c1 \notin \sigma(n) \wedge c2 \in \sigma(n)$ then $\chi_\sigma(c1, n) = \chi_\sigma(c1, n-1) \wedge \chi_\sigma(c2, n) = \chi_\sigma(c2, n-1) + 1$ then HR(n).
- If $c1 \in \sigma(n)$ then $c2 \in \sigma(n)$ and $\chi_\sigma(c1, n) = \chi_\sigma(c1, n-1) + 1 \wedge \chi_\sigma(c2, n) = \chi_\sigma(c2, n-1) + 1$ then HR(n)

Eq. 1a forbids the fourth case. ∎

From Proposition 3, we deduce that we need to build an arc in the CCG from c_2 to c_1 every time we find a constraint of the form $c_1 \boxed{\subset} c_2$. However, because this constraint is bounded (see Definition 13), we do not build any counter-arc in that case.

All the expressions based on Subclocking, *i.e.*,Union and Intersection, also imply some causality relations. Here again, the constraints are bounded relations and consequently, no counter-arc is added to the CCG. Let us show these implications.

Proposition 4 (Union and subclocking) *A clock is always a subclock of the union of itself with any other clock:* $\sigma \models_{ccsl} u \triangleq c_1 \boxed{+} c_2 \implies (\sigma \models_{ccsl} c_1 \boxed{\subset} u \wedge \sigma \models_{ccsl} c_2 \boxed{\subset} u)$.

Proof of Proposition 4 Let us assume $\sigma \models_{ccsl} u \triangleq c_1 \boxed{+} c_2$.
$(c_1 \in \sigma(n) \implies (c_1 \in \sigma(n) \vee c_2 \in \sigma(n)) \implies u \in \sigma(n)) \implies \sigma \models_{ccsl} c_1 \boxed{\subset} u$.
$(c_2 \in \sigma(n) \implies (c_1 \in \sigma(n) \vee c_2 \in \sigma(n)) \implies u \in \sigma(n)) \implies \sigma \models_{ccsl} c_2 \boxed{\subset} u$. ∎

Corollary 1 (Union and causality) *The union of two clocks is faster than both clocks:* $\sigma \models_{ccsl} u \triangleq c_1 \boxed{+} c_2 \implies (\sigma \models_{ccsl} u \boxed{\preccurlyeq} c_1 \wedge \sigma \models_{ccsl} u \boxed{\preccurlyeq} c_2)$.

The corollary comes directly from Propositions 3 and 4.

Proposition 5 (Intersection and subclocking) *The intersection of two clocks is a subclock of both clocks:* $\sigma \models_{ccsl} i \triangleq c_1 \boxed{*} c_2 \implies (\sigma \models_{ccsl} i \boxed{\subset} c_1 \wedge \sigma \models_{ccsl} i \boxed{\subset} c_2)$.

Proof of Proposition 5 Let us assume $\sigma \models_{ccsl} i \triangleq c_1 \boxed{*} c_2$.
$(i \in \sigma(n) \implies (c_1 \in \sigma(n) \wedge c_2 \in \sigma(n)) \implies c_1 \in \sigma(n)) \implies \sigma \models_{ccsl} i \boxed{\subset} c_1$.
$(i \in \sigma(n) \implies (c_1 \in \sigma(n) \wedge c_2 \in \sigma(n)) \implies c_2 \in \sigma(n)) \implies \sigma \models_{ccsl} i \boxed{\subset} c_2$. ∎

Corollary 2 (Intersection and causality) *The intersection of two clocks is slower than both clocks:* $\sigma \models_{ccsl} i \triangleq c_1 \boxed{*} c_2 \implies (\sigma \models_{ccsl} c_1 \boxed{\preccurlyeq} i \wedge \sigma \models_{ccsl} c_2 \boxed{\preccurlyeq} i)$.

Here again, the corollary comes directly from Propositions 3 and 5.

To be complete, one should also show that Union (resp. Intersection) does not imply any causality relations between the clocks themselves but only between the union clock u (resp. the intersection clock i) and the clocks c_1 and c_2. To do so, consider a schedule, where c_1 would tick alone. None of the binary relations can prevent c_1 from ticking and thus, the distance between c_1 and c_2 can grow infinitely large, thus preventing from having an upper bound. If now, we consider a schedule were c_2 ticks alone and c_1 never ticks, then such a schedule does not violate an union or intersection constraint and still prevents us from having a lower bound.

The next step is to determine what causality relations are implied by expressions Infimum and Supremum.

Proposition 6 (Infimum and causality) *The infimum of two clocks is always faster than both clocks:* $\sigma \models_{ccsl} inf \triangleq c_1 \boxed{\wedge} c_2 \implies (\sigma \models_{ccsl} inf \boxed{\preccurlyeq} c_1 \wedge \sigma \models_{ccsl} inf \boxed{\preccurlyeq} c_2)$.

Proof of Proposition 6 Let us assume $\sigma \models_{ccsl} inf \triangleq c_1 \boxed{\wedge} c_2$.
$(\chi_\sigma(inf, n) = max(\chi_\sigma(c_1, n), \chi_\sigma(c2, n)) \implies \chi_\sigma(inf, n) \geq \chi_\sigma(c_1, n)) \implies \sigma \models_{ccsl} inf \boxed{\preccurlyeq} c_1$.
Similarly, $\chi_\sigma(inf, n) \geq \chi_\sigma(c_2, n)) \implies \sigma \models_{ccsl} inf \boxed{\preccurlyeq} c_2$. ∎

Proposition 7 (Supremum and causality) *The supremum of two clocks is always slower than both clocks:* $\sigma \models_{ccsl} sup \triangleq c_1 \boxed{\vee} c_2 \implies (\sigma \models_{ccsl} c_1 \boxed{\preccurlyeq} sup \wedge \sigma \models_{ccsl} c_2 \boxed{\preccurlyeq} sup)$:

Proof of Proposition 7 Let us assume $\sigma \models_{ccsl} sup \triangleq c_1 \boxed{\vee} c_2$.
$(\chi_\sigma(sup, n) = min(\chi_\sigma(c_1, n), \chi_\sigma(c2, n)) \implies \chi_\sigma(c_1, n) \geq \chi_\sigma(sup, n)) \implies \sigma \models_{ccsl} c_1 \boxed{\preccurlyeq} sup$.
Similarly, $\chi_\sigma(c_2, n) \geq \chi_\sigma(sup, n)) \implies \sigma \models_{ccsl} c_2 \boxed{\preccurlyeq} sup$. ∎

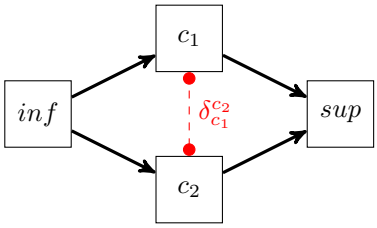


Figure 3: Causality Clock Graph for Infimum and Supremum.

The same reasoning as for the Union and Intersection can be used again to show that there is no causality relation between c_1 and c_2 imposed by either Infimum or Supremum. However, these binary expressions are unbounded (see Definition 14), then we need to add a counter-arc (c_1, c_2) in the CCG (see Figure 3). We know that inf is faster than both c_1 and c_2 but we need to bound the counter $\delta_{c_1}^{c_2}$ between c_1 and c_2. Similarly, we know that both c_1 and c_2 are faster than sup.

The last step is to consider the unary expression Delay.

Proposition 8 (Delay and causality) *A clock is always faster than any clock that is delayed from it:* $\forall d \in \mathbb{N}, \sigma \models_{ccsl} del \triangleq c \$ d \implies 0 \geq \delta_{del}^{c} \geq -d$

Proof of Proposition 8 If $\chi_\sigma(c, n) \leq d$ then Eq. 5 $\implies \chi_\sigma(del, n) = 0$. Otherwise, $\chi_\sigma(del, n) = \chi_\sigma(c, n) - d$. In both cases, $0 \geq \delta_{del}^{c} \geq -d$. ∎

From Proposition 8, we can deduce that we have both a lower and an upper bound, therefore we must add two arcs: one from c to del and one from del to c. Since the constraint is bounded, no counter-arc must be added in the CCG.

4 A state-based semantics for CCSL operators

This section gives a formal definition of CCSL operators in terms of labeled transition systems. Some of the CCSL operators require an infinite number of states.

4.1 CCSL clocks and relations

Definition 19 (Labeled Transition System) *A* Labeled Transition System *[9] over a set A of actions is defined as a tuple $\mathcal{A} = \langle S, T, s0, \alpha, \beta, \lambda \rangle$ where*

- *S is a set of* states*,*
- *T is a set of* transitions*,*
- *$s0 \in S$ is the initial state,*
- *$\alpha, \beta : T \to S$ denote respectively the* source state *and the* target state *of a transition,*
- *$\lambda : T \to A$ denotes the action responsible for a transition,*
- *the mappings $\langle \alpha, \lambda, \beta \rangle : T \to S \times A \times S$ are one-to-one so that T is a subset of $S \times A \times S$.*

In the context of CCSL, the actions are clocks. For each CCSL **clock** c, we build the Labeled Transition System $Clock_c = \langle S, T, \alpha, \beta, \lambda \rangle$ over $A_c = \{c, \epsilon\}$ such that

- $S = \{s\}$, $T = \{t, e\}$, $s0 = s$,
- $\alpha(t) = \alpha(e) = \beta(t) = \beta(e) = s$,
- $\lambda(t) = c$ and $\lambda(e) = \epsilon$.

The ϵ action allows for doing nothing. This is to allow composition with other LTSs. $Clock_a$ is given in Figure 4.a as an illustration.[2]

Definition 20 (Synchronization constraint) *Given n sets of actions A_1, ..., A_n, a* synchronization constraint *is a subset I of $A_1 \times \ldots \times A_n$.*

Definition 21 (Synchronized product) *If, for $i = 1, \ldots, n$, $\mathcal{A}_i = \langle S_i, T_i, s0_i, \alpha_i, \beta_i, \lambda_i \rangle$ is a labeled transition system over A_i, and if $I \subseteq A_1 \times \ldots \times A_n$ is a synchronization constraint, the* synchronized product *[9] of $\mathcal{A}_i$ with respect to I is the labeled transition system $\langle S, T, s0, \alpha, \beta, \lambda \rangle$ over the set I defined by*

- $S = S_1 \times \ldots \times S_n$, $s0 = s0_1 \times \ldots \times s0_n$,
- $T = \{\langle t_1, \ldots, t_n \rangle \in T_1 \times \ldots \times T_n | \langle \lambda_1(t_1), \ldots, \lambda_n(t_n) \rangle \in I\}$,
- $\alpha(\langle t_1, \ldots, t_n \rangle) = \langle \alpha_1(t_1), \ldots, \alpha_n(t_n) \rangle$,
- $\beta(\langle t_1, \ldots, t_n \rangle) = \langle \beta_1(t_1), \ldots, \beta_n(t_n) \rangle$,
- $\lambda(\langle t_1, \ldots, t_n \rangle) = \langle \lambda_1(t_1), \ldots, \lambda_n(t_n) \rangle$.

Synchronization constraints allow for capturing the semantics of CCSL polychronous operators. In this section, we focus on CCSL (binary) relations.

Relation 1 (Coincidence) *Given two clocks $c1$ and $c2$,* coincidence *$c1$ $=$ $c2$ is the synchronized product of $Clock_{c1}$ and $Clock_{c2}$ with respect to the synchronization constraint $I = \{\langle c1, c2 \rangle, \langle \epsilon, \epsilon \rangle\}$ (Fig. 4.b).*

Relation 2 (Subclocking) CCSL subclock *($c1$ $\subset$ $c2$) is the synchronized product of $Clock_{c1}$ and $Clock_{c2}$ with respect to the synchronization constraint $I = \{\langle c1, c2 \rangle, \langle \epsilon, c2 \rangle, \langle \epsilon, \epsilon \rangle\}$ (Fig. 4.c).*

Relation 3 (Exclusion) *Figure 4.d illustrates* CCSL excludes *($c1$ $\#$ $c2$) defined as the synchronized product of $Clock_{c1}$ and $Clock_{c2}$ with respect to the synchronization constraint $I = \{\langle c1, \epsilon \rangle, \langle \epsilon, c2 \rangle, \langle \epsilon, \epsilon \rangle\}$.*

4.2 CCSL bounded expressions

In CCSL, expressions allow for the creation of new clocks based on existing ones. Expressions can also be represented as labeled transition systems. Union and intersection are two simple examples of CCSL expressions.

Expression 1 (Union) $u \triangleq c1 + c2$ *(u is the* union *of $c1$ and $c2$) is represented by the synchronized product of $Clock_{c1}$, $Clock_{c2}$ and $Clock_u$ with respect to the synchronization constraint $I = \{\langle c1, c2, u \rangle, \langle c1, \epsilon, u \rangle, \langle \epsilon, c2, u \rangle, \langle \epsilon, \epsilon, \epsilon \rangle\}$ (Fig. 5.a).*

Expression 2 (Intersection) $i \triangleq c1.c2$ *(i is the* intersection *of $c1$ and $c2$) is represented by the synchronized product of $Clock_{c1}$, $Clock_{c2}$ and $Clock_i$ with respect to the synchronization constraint $I = \{\langle c1, c2, i \rangle, \langle c1, \epsilon, \epsilon \rangle, \langle \epsilon, c2, \epsilon \rangle, \langle \epsilon, \epsilon, \epsilon \rangle\}$ (Fig. 5.b).*

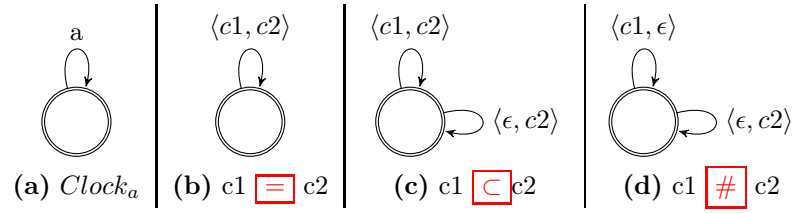
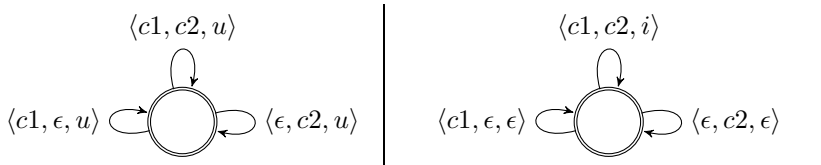


Figure 4: Primitive CCSL relations as Labeled Transition Systems



(a) u is the union of $c1$ and $c2$ (b) i is the intersection of $c1$ and $c2$

Figure 5: Union and intersection of clocks

Those two expressions are stateless (one state). Other expressions are stateful and require building dedicated LTS to express their semantics.

Expression 3 (Binary delay) *The* binary delay *(delayed* $\triangleq$ *base* \$ *n) is represented by a dedicated labeled transition system* $Delay(n) = \langle S, T, s0, \alpha, \beta, \lambda\rangle$ *over* $A = \{init, steady, \epsilon\}$ *with* $n+1$ *states such that*

- $S = \{d_0, d_1, \ldots, d_n\}$, $T = \{t_0, t_1, \ldots, t_n, e_0, \ldots, e_n\}$, $s0 = d_0$,
- $\alpha(t_i) = d_i$ *and* $\alpha(e_i) = d_i$ *for* $i \in \{0 \ldots n\}$,
- $\beta(t_i) = d_{i+1}$ *for* $i \in \{0 \ldots n\}$ *and* $\beta(t_n) = d_n$,
  $\beta(e_i) = d_i$ *for* $i \in \{0 \ldots n\}$,
- $\lambda(t_i) = init$ *for* $i \in \{0 \ldots n-1\}$ *and* $\lambda(t_n) = steady$ *and* $\lambda(e_i) = \epsilon$ *for* $i \in \{0 \ldots n\}$.

init denotes a preliminary phase during which the *base* clock must tick alone. *steady* is a phase where both clocks *base* and *delayed* become synchronous for ever. Figure 6 gives as an illustration the resulting transition systems to denote $b \triangleq a$ \$ 1 (actions *init* and *steady* are hidden).

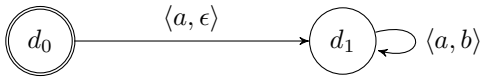


Figure 6: Binary delay: $b \triangleq a$ \$ 1

The binary delay is a particular case of a more general synchronous expression called $FilteredBy$ (denoted ▼). $f \triangleq c$ ▼ $u.(v)^\omega$ defines the clock f as a subclock of c according to two binary words u and v.

[2]The ϵ transitions are not shown to simplify the drawings. In all the presented LTSs, it is always possible to do nothing by remaining in the same state.

Definition 22 (Binary word) *A* binary word w *is a function,* $w : \mathbb{N}_{>0} \to \{0, 1, \bot\}$*, such that* $(\exists l \in \mathbb{N}_{>0}, w(l) = \bot) \implies ((\forall i > l)(w(i) = \bot))$.

Definition 23 (Length of a binary word) *If* w *is a binary word,* $len(w)$ *(denoted* $|w|$*) is called its* length. $len : (\mathbb{N}_{>0} \to \{0, 1, \bot\}) \to \mathbb{N} \cup \{\omega\}$. *If* $\forall i \in \mathbb{N}_{>0}, w(i) \neq \bot$ *then* $|w| = \omega$ *and* w *is said to be an* infinite word*, otherwise* w *is a* finite word. *When* w *is finite,* $|w| = min(i \in \mathbb{N}, w(i + 1) = \bot)$.

Definition 24 (Exponentiation of a binary word) *Let* n *be a positive natural number (*$n \in \mathbb{N}_{>0}$*). Let* v *be a finite binary word.* $w = v^n$ *is a finite binary word such that* $|w| = n * |v|$ *and* $\forall i \in 1..n, \forall j \in \{1..|v|\}, w(i * j) = v(j)$.

Definition 25 (Infinitely periodic binary word) *Let* v *be a finite binary word.* $w = (v)^\omega$ *is an infinite binary word such that* $\forall i \in \mathbb{N}, \forall j \in \{1..|v|\}, w(i * |v| + j) = v(j)$.

Definition 26 (Concatenation of binary words) *Let* u *and* v *be two binary words,* u *is finite.* $w = u.v$ *is a binary word such that* $(i \leq |u| \implies w(i) = u(i)) \wedge (i > |u| \implies w(i) = v(i - |u|))$, $\forall i \in \mathbb{N}_{>0}$. *If* v *is infinite, then* w *is infinite. If* v *is finite, then* w *is finite and such that* $|w| = |u| + |v|$.

Expression 4 (Filtering) *If* u *and* v *are two finite binary words, the LTS for* CCSL *expression* FilteredBy *is defined as follows.* $f \triangleq c \blacktriangledown u.(v)^\omega$ *is the LTS* $Filter(u, v) = \langle S, T, s0, \alpha, \beta, \lambda \rangle$ *over* $A = \{zero, one, \epsilon\}$ *with* $n + 1$ *states s.t.,*

- $S = \{s_1, \ldots, s_{|u|+|v|}\}$, $T = \{t_1, \ldots, t_{|u|+|v|}, e_1, \ldots, e_{|u|+|v|}\}$, $s0 = s_1$,
- $\alpha(t_i) = s_i$ *for* $i \in \{1 \ldots |u| + |v|\}$,
- $\beta(t_i) = s_{i+1}$ *for* $i \in \{1 \ldots |u| + |v| - 1\}$ *and* $\beta(t_{|u|+|v|}) = s_{|u|+1}$,
- $\lambda(t_i) = zero$ *if* $u(i) = 0$ *and* $\lambda(t_i) = one$ *if* $u(i) = 1$*, for* $i \in \{1 \ldots |u|\}$
- $\lambda(t_{i+|u|}) = zero$ *if* $v(i) = 0$ *and* $\lambda(t_{i+|u|}) = one$ *if* $v(i) = 1$*, for* $i \in \{1 \ldots |v|\}$
- $\alpha(e_i) = s_i$ *and* $\beta(e_i) = s_i$ *and* $\lambda(e_i) = \epsilon$ *for* $i \in \{1 \ldots |u| + |v|\}$.

The label *one* denotes instants where both f and c tick together. The label *zero* when c ticks alone. Actually, Delay is just a particular case of filter with $u = 0^n$ and $v = 1$. Another interesting special case is when $u = 0^d$ and $v = 1.0^{p-1}$, for $d \in \mathbb{N}_{>0}$ and $p \in \mathbb{N}$. This defines a **periodic pattern** $Periodic(d, p)$, where d is called the *offset* and p the *period*. $Delay(n)$ is also a particular periodic case with an offset of n and a period of 1.

Figure 7 gives an example of a periodic filter, where b is periodic on a with a period of 3 and an offset of 1: $b \triangleq a \blacktriangledown 0.(1.0.0)^\omega$.

Expression 5 (Sampling) $sampled \triangleq trigger$ ***sampledOn*** $base$ *is the LTS* $Sampled = \langle S, T, s0, \alpha, \beta, \lambda \rangle$ *over* $A = \{base, trig, sample, all\epsilon\}$ *with* 2 *states such that,*

- $S = \{s_1, s_2\}$, $T = \{b, bs, sa_1, sa_2, t_1, t_2, e_1, e_2\}$, $s0 = s_1$,
- $\alpha(b) = \beta(b) = s_1$ *and* $\lambda(b) = base$,
- $\alpha(sa_i) = \beta(sa_i) = s_i$ *and* $\lambda(sa_i) = all$ *for* $i \in \{1 \ldots 2\}$,
- $\alpha(t_i) = s_i$ *and* $\beta(t_i) = s_2$ *and* $\lambda(t_i) = trig$ *for* $i \in \{1 \ldots 2\}$,

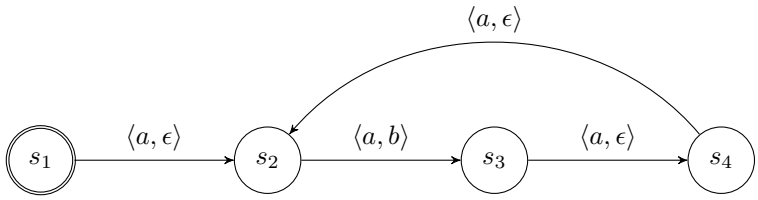


Figure 7: Example of periodic filter with offset: $b \triangleq a \blacktriangledown 0.(1.0.0)^{\omega}$

- $\alpha(bs) = s_2$ *and* $\beta(bs) = s_1$ *and* $\lambda(bs) = sample$,
- $\alpha(e_i) = \beta(e_i) = s_i$ *and* $\lambda(e_i) = \epsilon$ *for* $i \in \{1 \dots 2\}$.

SampledOn is an expression that produces a clock s if and only if a *trigger* has ticked since the previous tick of a sampling clock (*base*). Labels *base* and *trig* respectively denote instants where clocks *base* and *trigger* tick alone. Label *sample* denotes instants where both clocks *base* and *sampled* tick simultaneously. Label *all* denotes instants where all the three clocks *base*, *trigger* and *sampled* tick simultaneously.

Figure 8 gives the LTS for the sampling operator.

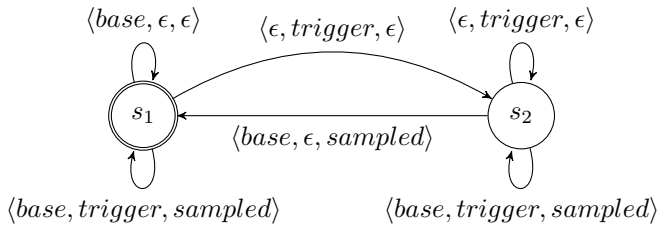


Figure 8: Sampling: $sampled \triangleq trigger$ **sampledOn** *base*

4.3 Unbounded relations

Unbounded operators can be modeled with labeled transition systems that have an infinite but countable number of states.

Relation 4 (Precedence) Precedence $left \prec right$ *is a labeled transition system* $Precedes = \langle S, T, s0, \alpha, \beta, \lambda \rangle$ *over* $A = \{left, right, both, \epsilon\}$ *s.t.,*

- $S = \{p_i | i \in \mathbb{N}\}$, $T = \{l_i, r_i, lr_i, e_i | i \in \mathbb{N}\}$, $s0 = p_i$,
- $\alpha(l_i) = \alpha(e_i) = \alpha(lr_i) = p_i \wedge \alpha(r_i) = p_{i+1}, \forall i \in \mathbb{N}$,
- $\beta(l_i) = p_{i+1} \wedge \beta(r_i) = \beta(e_i) = \beta(lr_i) = p_i, \forall i \in \mathbb{N}$,
- $\lambda(l_i) = left \wedge \lambda(r_i) = right \wedge \lambda(lr_i) = both \wedge \lambda(e_i) = \epsilon, \forall i \in \mathbb{N}$.

Label $left$ denotes instants where clock $left$ must tick alone. Label $right$ denotes instants where clock $right$ must tick alone. Label *both* denotes instants where the two clocks must

tick simultaneously. Figure 9 shows the transition system for the CCSL relation $a \prec b$, *i.e.,* the synchronized product of $Clock_a$, $Clock_b$ and *Precedes* with respect to the synchronization constraint $I = \{\langle a, \epsilon, left\rangle, \langle \epsilon, b, right\rangle, \langle a, b, both\rangle, \langle \epsilon, \epsilon, \epsilon\rangle\}$ ($left$, $right$ and $both$ are hidden for the sake of simplicity).

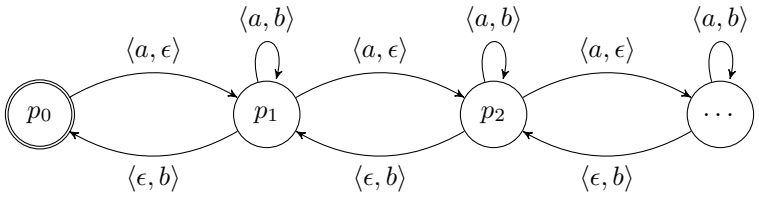


Figure 9: CCSL precedence (infinite state LTS): a precedes b.

This operator is called *unbounded* because the drift between a and b is not bounded, *i.e.,* a can tick infinitely often without b ticking at all. This operator is not symmetrical. Even though a is unconstrained, b on the contrary is constrained to be always a little late compared to a. b is said to be slower than a, or a is faster than b.

4.4 Unbounded expressions

In CCSL, there are two unbounded expressions that constrain neither a nor b: **Inf** and **Sup**.

Expression 6 (Infimum) *$Inf(a, b)$ is the labeled transition system $Inf = \langle S, T, s0, \alpha, \beta, \lambda\rangle$ over $A = \{left, right, both, left\_inf, right\_inf, \epsilon\}$ such that*

- $S = \{s_i | i \in \mathbb{Z}\}$, $T = \{inc_i, dec_i, t_i, e_i | i \in \mathbb{Z}\}$, $s0 = s_0$,
- $\alpha(inc_i) = \alpha(dec_i) = \alpha(both_i) = \alpha(e_i) = s_i$, $\forall i \in \mathbb{Z}$,
- $\beta(both_i) = \beta(e_i) = s_i$ *and* $\beta(inc_i) = s_{i+1}$ *and* $\beta(dec_i) = s_{i-1}$, $\forall i \in \mathbb{Z}$,
- $\lambda(inc_i) = left\_inf$ *if* $i \geq 0$, *and* $\lambda(inc_i) = left$ *if* $i < 0$, $\forall i \in \mathbb{Z}$
- $\lambda(dec_i) = right\_inf$ *if* $i \leq 0$, *and* $\lambda(dec_i) = right$ *if* $i < 0$, $\forall i \in \mathbb{Z}$
- $\lambda(both_i) = both$ *and* $\lambda(e_i) = \epsilon$, $\forall i \in \mathbb{Z}$

Inf(a, b) is the slowest clock that is faster than both a and b. In most cases, $Inf(a, b)$ is neither a nor b but a clock that sometimes tick simultaneously with a (when a is in advance over b), sometimes it ticks simultaneously with b (when a is late compared to b) and sometimes it ticks simultaneously with a and b (when none of them precedes the other one). Figure 10 shows the transition systems for $i \triangleq Inf(a, b)$. This LTS is infinite on both sides. By definition $Inf(a, b) \preccurlyeq a$ and $Inf(a, b) \preccurlyeq b$, which means that if $Inf(a, b)$ is somehow constrained (*i.e.,* by a synchronous operator like filter), then this propagates the constraint on both a and b. Additionally, the tickings of $Inf(a, b)$ are constrained (and bounded) by all the clocks faster than either a or b.

Expression 7 (Supremum) *$Sup(a, b)$ is a labeled transition system $Sup = \langle S, T, s0, \alpha, \beta, \lambda\rangle$ over $A = \{left, right, both, left\_sup, right\_sup, both\_sup, \epsilon\}$ such that*

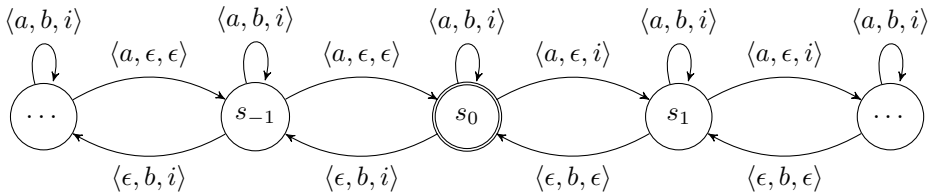


Figure 10: CCSL Inf (infinite state LTS): $i \triangleq Inf(a, b)$.

- $S = \{s_i | i \in \mathbb{Z}\}$, $T = \{inc_i, dec_i, t_i, e_i | i \in \mathbb{Z}\}$, $s0 = s_0$,
- $\alpha(inc_i) = \alpha(dec_i) = \alpha(both_i) = \alpha(e_i) = s_i$, $\forall i \in \mathbb{Z}$,
- $\beta(both_i) = \beta(e_i) = s_i$ *and* $\beta(inc_i) = s_{i+1}$ *and* $\beta(dec_i) = s_{i-1}$, $\forall i \in \mathbb{Z}$,
- $\lambda(inc_i) = left$ *if* $i \geq 0$, *and* $\lambda(inc_i) = left\_sup$ *if* $i < 0$, $\forall i \in \mathbb{Z}$
- $\lambda(dec_i) = right$ *if* $i \leq 0$, *and* $\lambda(dec_i) = right\_sup$ *if* $i < 0$, $\forall i \in \mathbb{Z}$
- $\lambda(both_i) = both$ *if* $i \neq 0$ *and* $\lambda(e_i) = \epsilon$, $\forall i \in \mathbb{Z}$, *and* $\lambda(both_0) = both\_sup$

Sup(a, b) is defined as the fastest clock that is slower than both a and b. In most cases, $Sup(a, b)$ is neither a nor b. Figure 11 shows the transition systems for $s \triangleq Sup(a, b)$. By definition $a \preccurlyeq Sup(a, b)$ and $b \preccurlyeq Sup(a, b)$, which means that the constraints imposed on $Sup(a, b)$ do not directly impact neither a or b. However, whenever a clock c is known to be slower than either a or b, then it is also slower than $Sup(a, b)$, *i.e.,* ($\exists c$ such that $a \preccurlyeq c \vee b \preccurlyeq c$) $\implies Sup(a, b) \preccurlyeq c$.

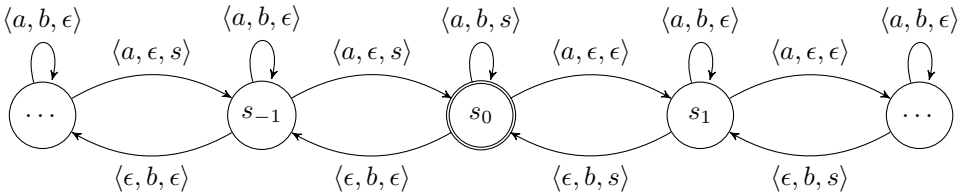


Figure 11: CCSL sup (infinite state LTS): $s \triangleq Sup(a, b)$.

5 Boundness issues on CCSL specifications

When several CCSL constraints are put in parallel, the composition is defined as the synchronized product of the LTSs of the operators. However, since some of the LTSs for the primitive operators are infinite (*e.g.,* Relation 4, or Expressions 6-7), the synchronized product might end up being infinite. However, even though the product is potentially infinite, in some cases, only a finite subset of the synchronized product is reachable from the initial state. We show a case where the product of infinite LTSs is finite. The algorithm used in that subsection only terminates when the product is actually finite.

Considering n LTSs such that, for $i = 1, \ldots, n$, $\mathcal{A}_i = \langle S_i, T_i, s0_i, \alpha_i, \beta_i, \lambda_i \rangle$ and one synchronization constraint $I \subseteq A_1 \times \ldots \times A_n$, the synchronized product of $\mathcal{A}_i$ with respect to I is a labeled transition system $\langle S, T, s0, \alpha, \beta, \lambda \rangle$ over the set I constructed as described in Algorithm 1.

Algorithm 1 Synchronized product through reachability analysis
Let $S \leftarrow \emptyset$, $T \leftarrow \emptyset$,
Let $s0 \leftarrow s0_1 \times \ldots \times s0_n$
Let $S' \leftarrow \{s0\}$
while S' is not empty {
  Let $st = st_1 \times \ldots \times st_n$ *be one element of* S'
  Let $S \leftarrow S \cup \{st\}$
  Let $S' \leftarrow S' \setminus \{st\}$
  $\forall t = \langle t_1, \ldots, t_n \rangle \in T_1 \times \ldots \times T_n$ *such that*
    $(\forall i \in \{1 \ldots n\})(\alpha_i(t_i) = st_i)$ *and* $\lambda_1(t_1) \times \ldots \times \lambda_n(t_n) \in I$ *{*
    Let $st' = \beta_1(t_1) \times \ldots \times \beta_n(t_n)$
    if $st' \notin S$ *then* $S' \leftarrow S' \cup \{st'\}$
    $T \leftarrow T \cup \{t\}$, $\alpha(t) = st$, $\beta(t) = st'$, $\lambda(t) = \lambda_1(t_1) \times \ldots \times \lambda_n(t_n)$,
  }
}

Theorem 5.1 *Algorithm 1 terminates if and only if the product has a finite number of states.*

Proof S' is initialized with one state. At each iteration, one state st is removed from S' and added to S. All the outgoing transitions of st are computed. If C is the set of clocks, there are at most $2^{|C|}$ outgoing transitions. Some of these transitions may be inconsistent. For each transition the target state st' is computed and added to S' if not already present in S. This condition guarantees that the same state is not visited twice. The algorithm terminates when S' is empty. S' becomes empty when all the targeted state are already in S (have already been visited). If the set of reachable states is finite then when all the states are in S then S' is necessarily empty. Therefore, when the set of reachable states is finite the algorithm terminates. If there is an infinite number of reachable states, then S' is never empty and the algorithm never terminates. ∎

Let us take as an example the following CCSL specification: $(a \boxed{\prec} b) \wedge (a' \triangleq a \$ 1) \wedge (b \boxed{\prec} a')$. This specification is defined as the synchronized product of *Precedes* (Relation 4), *Delay*(1) (Expression 3), *Precedes* (Relation 4 again).

Initially, $s0 = p_0 \times d_0 \times p_0$. The first *precedes* (state p_0) imposes b not to tick, the second *precedes* (state p_0) prevents a' from ticking whereas the *delay* (state d_0) only allows a to tick alone without a'. Therefore the only outgoing transition consists in making a ticks alone going into the state $s1 = p_1 \times d_1 \times p_0$. At this stage $S' = \{s1\}$ and $S = \{s0\}$. From $s1$, the first *precedes* (state p_1) does not impose any constraint while the second one (state p_0) still prevents a' from ticking. The *delay* (state $d1$) only allows making a and a' tick simultaneously. Since a' cannot tick, then a cannot tick either, so only b can tick leading to state $s2 = p_0 \times d_1 \times p_1$. Therefore $S = \{s0, s1\}$ and $S' = \{s2\}$. From $s2$, the first *precedes* prevents b from ticking, the second relation also prevents b from ticking. The *delay* only allows a and a' to tick simultaneously. Taking this (sole) solution leads to $s1$, which is already in S, so no new state is added to S'. S' being therefore empty, the algorithm terminates with $S = \{s0, s1, s2\}$ (Fig. 12).

This particular construction is very frequent, it has been called **Alternation** and is denoted $a \boxed{\sim} b$. Increasing the delay from 1 to n makes a particular relation, called **bounded precedence** and denoted as $a \boxed{\prec_n} b$: $a \boxed{\sim} b \equiv a \boxed{\prec_1} b$. Previous works on CCSL were always

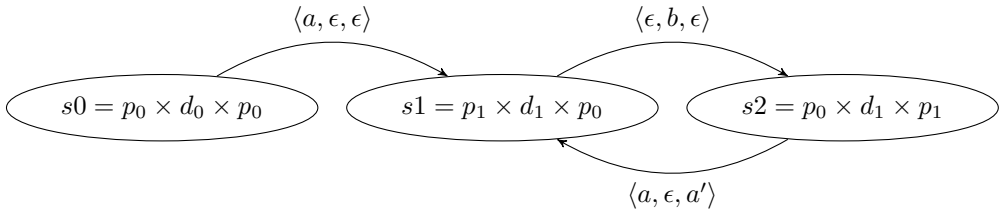


Figure 12: CCSL alternation: synchronized product of two precedences and one delay

assuming a bound for all CCSL operators, whereas here the bound is computed by reachability analysis. However, the (semi) algorithm sketched above may not terminate when the synchronized product is not finite.

6 Example: CCSL for capturing the architecture, application and allocation

To illustrate the approach, we take an example inspired by [10], that was used for flow latency analysis on AADL[3] specifications [11]. However, with CCSL we are conducting different kinds of analyses, section 7 discusses some common points with classical real-time scheduling analysis.

6.1 Application

Figure 13 (on the top) considers a simple application described as a UML structured class. This application captures two inputs $in1$ and $in2$, performs some calculations ($step1$, $step2$ and $step3$) and then produces a result *out*. This application has the possibility to compute $step1$ and $step2$ concurrently depending on the chosen execution platform. This application runs in a streaming-like fashion by continuously capturing new inputs and producing outputs.

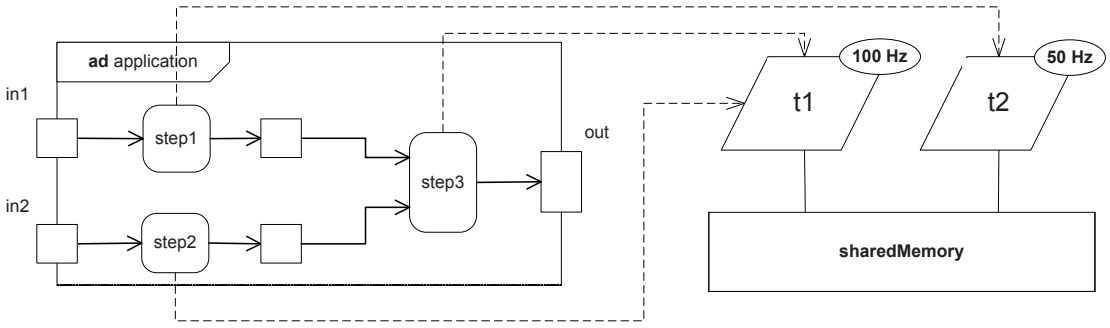


Figure 13: Simple application

To abstract this application as a CCSL specification, we assign one clock to each action. The clock has the exact same name as the associated action (*e.g., step*1). We also associate one clock

[3]AADL stands for Architecture & Analysis Description Language

with each input, this represents the capturing time of the inputs, and one clock with the production of the output (*out*). The successive instants of the clocks represent successive executions of the actions or input sensing time or output release time. The basic CCSL specification is:

$$in1 \boxed{\preccurlyeq} step1 \wedge step1 \boxed{\prec} step3 \tag{6}$$

$$in2 \boxed{\preccurlyeq} step2 \wedge step2 \boxed{\prec} step3 \tag{7}$$

$$step3 \boxed{\preccurlyeq} out \tag{8}$$

Eq. 6 specifies that *step*1 may begin as soon as an input *in*1 is available. Executing *step*3 also requires *step*1 to have produced its output. Eq. 7 is similar for *in*2 and *step*2. Eq. 8 states that an output can be produced as soon as *step*3 has executed. Note that CCSL precedence is well adapted to capture infinite FIFOs denoted on the figure as object nodes. Such a specification is clearly unbounded, therefore TimeSquare cannot perform any kind of exhaustive analysis and can only produce a particular schedule that matches the specification (see Fig. 14).

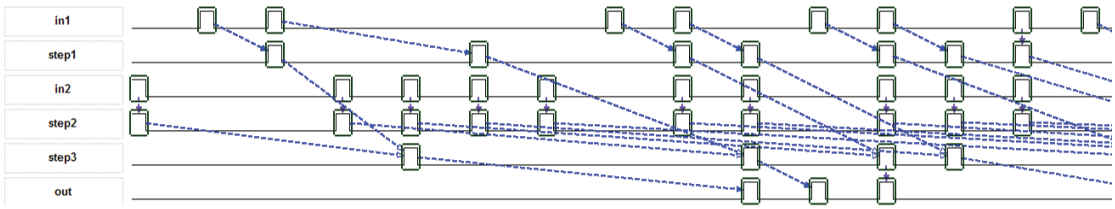


Figure 14: A valid schedule for the application part of Fig. 13

One way to reduce the state-space is to bound the drift between the inputs and the outputs. This means limiting the parallelism by slowing down the production of outputs when several computations are still on-going. This can easily be done by adding a CCSL constraint like Eq. 9.

$$Sup(in1, in2) \boxed{\sim} out \tag{9}$$

The effect of this constraint can be seen on Figure 15. Looking carefully at this schedule, we can note that the arrival of *in*2 has been slown down to avoid large accumulation of computations. For instance, the third occurrence of *in*2 is delayed after the second occurrence of *out*. However, we can see that the input *in*1 keeps arriving at a fast rate allowing executions of *step*1. However, the execution of *step*3 is stalled after the corresponding occurrence of *in*2 has been dealt with by *step*2 as required by Eq. 7.

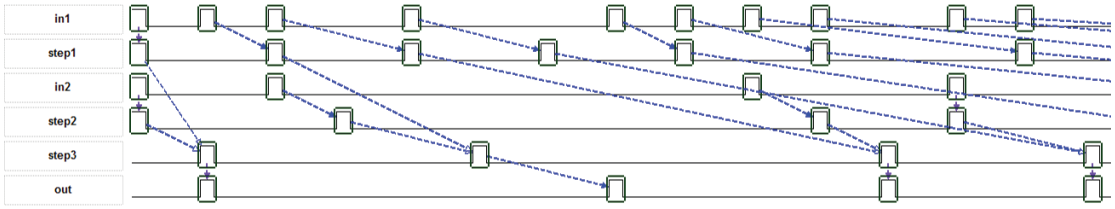


Figure 15: Another valid schedule for the application part of Fig. 13

Reachability analysis as described in Section 5 tells us that the composition is still not bounded because bounds on $Sup(in1, in2)$ do not imply bounds on both *in*1 and *in*2. To have

a complete finite systems, we can for instance replace Eq. 9 by Eq. 10.

$$Inf(in1, in2) \boxed{\sim} out \tag{10}$$

By doing so, our reachability analysis algorithm converges and produces a bounded state-space shown in Figure 16 [4]. We have removed $in1$, $in2$, and out since they were just adding interleaving without offering more actual parallelism in the execution of actions.

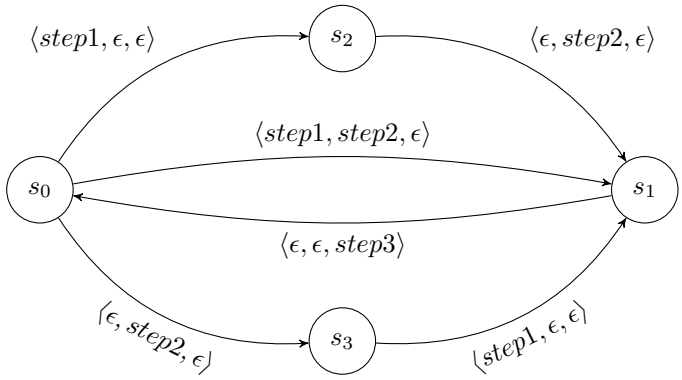


Figure 16: Synchronized product of Eqs. 6-8 and Eq. 10.

This kind of analysis is useful to detect invalid CCSL specifications. For instance, had we replaced Eq. 9 by Eq. 11 instead of Eq. 10, we would have obtained a finite result but with a typical case of deadlock in CCSL. Indeed, if from the initial state s_0, we decide to fire $in1$ (resp. $in2$) alone, then Eq. 11 prevents $in1+in2$ from ticking again before out ticks. But since $in2$ (resp. $in1$) was not produced and therefore $step2$ was not executed, then $step3$ cannot execute either since it requires both $step1$ and $step2$. If $step3$ cannot execute, then out cannot be produced, which then results in a deadlock.

$$in1 + in2 \boxed{\sim} out \tag{11}$$

6.2 Execution platform and allocation

Once the application is designed, then CCSL can also be used to capture the execution platform. Figure 13 (bottom part) shows the selected execution platform: two tasks with different activation periods. The basic CCSL specification of the execution platform is given as follows:

$$t1 \triangleq ms \blacktriangledown (1.0^9)^\omega \tag{12}$$

$$t2 \triangleq t1 \blacktriangledown (1.0)^\omega \tag{13}$$

Eq. 13 is a pure logical relationship between $t1$ and $t2$ that states that thread $t2$ is twice slower than thread $t1$, *i.e.,* it is periodic on $t1$ with period 2 and offset 0. Eq. 12 is also a periodic relation, but relative to ms, a particular clock that denotes milliseconds. Being periodic on ms with a period of 10 makes $t1$ a 100 Hz clock and therefore $t2$ a 50 Hz clock.

When the execution platform is specified, the remaining task is to map the application onto the execution platform. In MARTE, this is done through an allocation. In CCSL, this is done by

[4] The algorithm is available as an Eclipse update site on http://timesquare.inria.fr/sts/update_site/

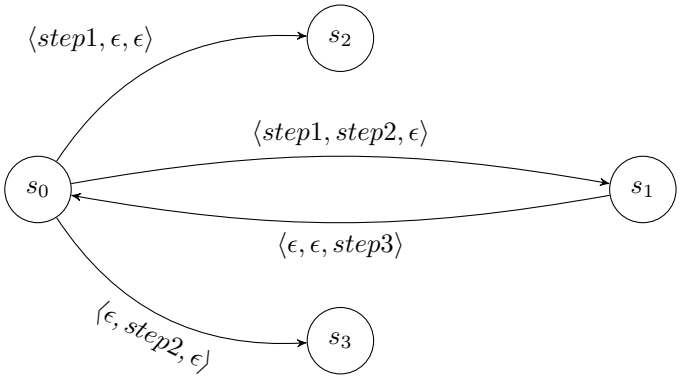


Figure 17: Synchronous products of Eqs. 6-8 and Eq. 11.

refining the two specifications with new constraints that specify this allocation. Since both *step*2 and *step*3 are allocated on the same thread, then their execution is exclusive (Eq. 14). Then, the thread being periodic, the inputs are sampled according to the period of activation of the threads (Eqs. 15-16). Then *step*3 needs inputs from both *step*1 and *step*2 before executing but it can execute only according to the sampling period of $t1$ since *step*3 is allocated to $t1$ (Eq. 17). Finally, all steps can only execute when their input data have been sampled (Eq. 18).

$$step2 \boxed{\#} step3 \tag{14}$$

$$in1\_s \triangleq in1 \textbf{ sampledOn } t1 \tag{15}$$

$$in2\_s \triangleq in2 \textbf{ sampledOn } t2 \tag{16}$$

$$d3\_s \triangleq \textbf{Inf}(step1, step2) \textbf{ sampledOn } t1 \tag{17}$$

$$in1\_s \boxed{\preccurlyeq} step1 \wedge in2\_s \boxed{\preccurlyeq} step2 \wedge d3\_s \boxed{\preccurlyeq} step3 \tag{18}$$

All these new constraints do not change anything on the finiteness of the whole system. They only reduce the set of possible executions. If the application specification was finite, then its allocated version is still finite. If it was infinite, they it remains infinite. Whether it is finite or not, TimeSquare can produce an execution of this specification (see Fig. 18). On this schedule the dashed arrows denote precedence relations, while the (red) vertical lines denote coincidence relations. Note that the fact that *ms* is a physical clock does not impact the calculus, it only impacts the visual representation of the schedule.

7 Related work

The transformation of CCSL into labeled transition systems has already been attempted in [12, 13]. However, in those attempts, the CCSL operators were bounded because the underlying model-checkers cannot deal with infinite labeled transition systems. The purpose of this work is to deal with unbounded operators.

In [14], there was an initial attempt to provide a data structure suitable to capture infinite transition systems based on a lazy evaluation technique. A similar structure could be used in our case except that we consider clocks with only two states (instead of three): tick or stall. Clock death is still to be further explored.

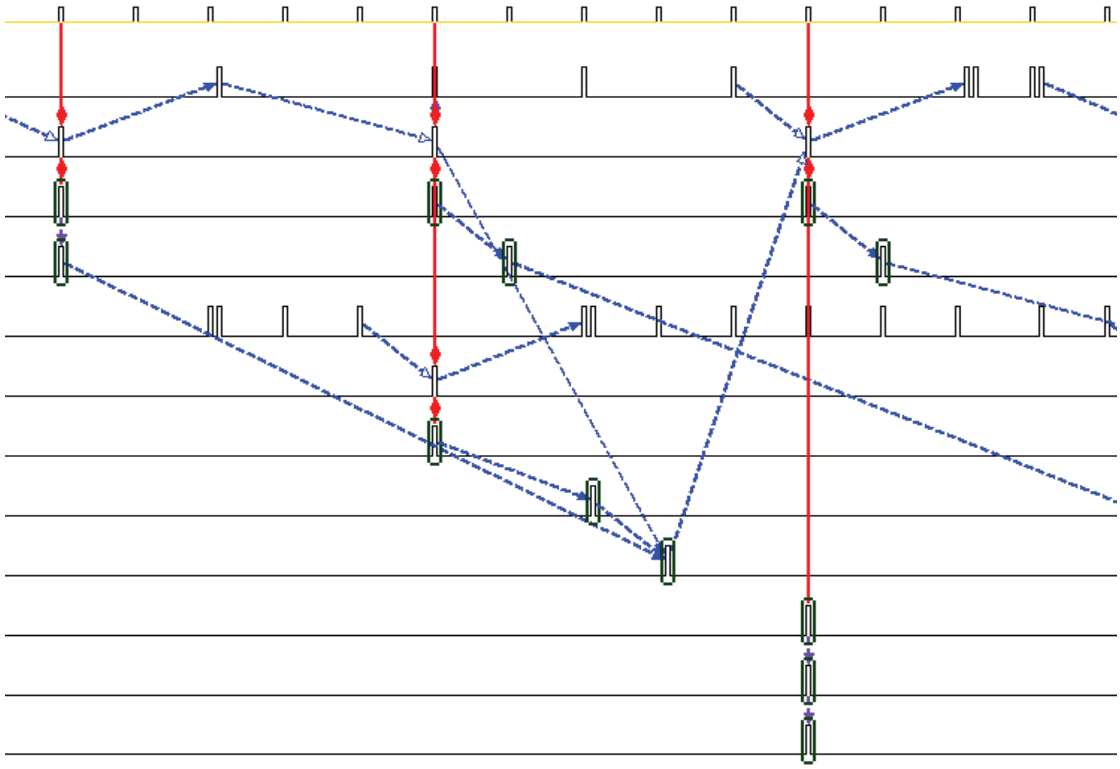


Figure 18: A valid schedule for the allocated application (Fig. 13)

The kind of applications addressed in section 6 is very close to models usually used in real-time scheduling theories. However, such theories usually rely on task models that abstract real applications. Originally they were rather simple (e.g., independent periodic tasks only for Rate Monotonic Analysis). Always more sophisticated models now appear in the literature. They are all based on numerous distinct parameters, providing numerical constraint values for timing aspects (dispatch time, period, deadline, jitter drift...). Tasks are considered as iterations of jobs (or jobs as instances of tasks). In our view, the successive timing values for characteristic feature of successive jobs can each be seen as a logical clock, and the time constraint relations between such clocks are usually expressed as simple equalities and bounded inequalities that fall well into the range of CCSL constructs descriptive power.

Classical (non real-time) scheduling, on its side, provides generally models where the initial constraints are less on timing and more on dependencies or on exclusive resource allocation. But resulting schedules are almost always of *modulo* periodic nature, here again matching the CCSL expressiveness.

Usually, authors [15, 16, 17] rely on "physical-by-nature" timing, found in theoretical models such as Timed Automata [18]. The distinctive difference is that timed automata assume a global physical time. Timed events are then constrained by value relations between so-called clocks (a different notion from our logical clocks), which are devices measuring physical time as it elapses.

Our work also bears some similarity with previous attempts by Alur and Weiss [19, 20], which define schedules as infinite words expressed in regular expressions and then construct corresponding Büchi automata.

8 Conclusion

We have presented a state-based semantics of a kernel subset of CCSL, a language that relies on logical clocks to express logical and temporal constraints. Each CCSL operator (relation or expression) is defined as a label transition system, that may have either a finite or infinite number of states. The parallel composition of CCSL constraints is defined as the synchronized product of the primitive label transition systems. A (semi)algorithm is proposed to actually build the synchronized product of infinite transition systems by assuming that only a finite number of states are accessible in the product. The algorithm only terminates on that condition. The work presented here improves on previous attempts to support exhaustive analyses of CCSL specifications. Indeed, previous works were only considering *a priori* bounded CCSL operators to guarantee the finiteness of the composition, while here no assumption is made on the boundness of primitive operators.

As a future work, we should extend and prove that data flow process networks can actually be used to detect finite compositions of any unbounded CCSL operators. Whereas it is pretty much clear that synchronous operators and regular asynchronous operators (like precedes, inf, sup) are always covered by synchronous data flow graphs, it is much less clear for mix operators like sampledOn.

References

[1] OMG: UML Profile for MARTE, v1.0. Object Management Group. (November 2009) formal/2009-11-02.

[2] André, C., Mallet, F., de Simone, R.: Modeling time(s). In: 10th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS '07). Number 4735 in LNCS, Nashville, TN, USA, ACM-IEEE, Springer (September 2007) 559–573

[3] André, C.: Syntax and semantics of the Clock Constraint Specification Language (CCSL). Research Report 6925, INRIA (May 2009)

[4] Deantoni, J., Mallet, F.: Timesquare: Treat your models with logical time. In: TOOLS (50). Volume 7304 of LNCS., Springer (2012) 34–41

[5] Benveniste, A., Caspi, P., Edwards, S.A., Halbwachs, N., Le Guernic, P., de Simone, R.: The synchronous languages 12 years later. Proc. of the IEEE **91**(1) (2003) 64–83

[6] Le Guernic, P., Talpin, J.P., Le Lann, J.C.: Polychrony for system design. Journal of Circuits, Systems, and Computers **12**(3) (2003) 261–304

[7] Lee, E.A., Sangiovanni-Vincentelli, A.L.: A framework for comparing models of computation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **17**(12) (December 1998) 1217–1229

[8] Mallet, F., Millo, J.V.: Boundness issues in CCSL specifications. In: ICFEM. Lecture Notes in Computer Science, Springer (2013) to appear.

[9] Arnold, A.: Finite transition systems - semantics of communicating systems. Int. Series in Computer Science. Prentice Hall (1994)

[10] Feiler, P.H., Hansson, J.: Flow latency analysis with the architecture analysis and design language. Technical Report CMU/SEI-2007-TN-010, CMU (June 2007)

[11] of Automotive Engineers, S.: SAE Architecture Analysis and Design Language (AADL). (June 2006) document number: AS5506/1.

[12] Yin, L., Mallet, F., Liu, J.: Verification of MARTE/CCSL time requirements in Promela/SPIN. In: ICECCS, IEEE Computer Society (2011) 65–74

[13] Gascon, R., Mallet, F., DeAntoni, J.: Logical time and temporal logics: Comparing UML MARTE/CCSL and PSL. In Combi, C., Leucker, M., Wolter, F., eds.: TIME, IEEE (2011) 141–148

[14] Romenska, Y., Mallet, F.: Lazy parallel synchronous composition of infinite transition systems. In: ICTERI. Volume 1000 of CEUR Workshop Proc. (2013) 130–145

[15] Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: Times: A tool for schedulability analysis and code generation of real-time systems. In: Formal Modeling and Analysis of Timed Systems. Volume 2791 of LNCS. Springer (2004) 60–72

[16] Krcál, P., Yi, W.: Decidable and undecidable problems in schedulability analysis using timed automata. In: Tools and Algorithms for the Construction and Analysis of Systems. Volume 2988 of LNCS. Springer (2004) 236–250

[17] Abdeddaim, Y., Asarin, E., Maler, O.: Scheduling with timed automata. Theoretical Computer Science **354**(2) (2006) 272–300

[18] Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2) (1994) 183–235

[19] Alur, R., Weiss, G.: Regular specifications of resource requirements for embedded control software. In: IEEE Real-Time and Embedded Technology and Applications Symp., IEEE CS (2008) 159–168

[20] Alur, R., Weiss, G.: Rtcomposer:a framework for real-time components with scheduling interfaces. In: Int. Conf. on Embedded software. EMSOFT '08, ACM (2008) 159–168

Contents

1 Introduction **3**

2 The Clock Constraint Specification Language **3**

3 Definitions **4**
3.1 Logical time model 4
3.1.1 Clock relations 5
3.1.2 Clock definitions 6
3.2 Composition 6
3.3 Safety issues 8
3.4 Building the causality clock graph 9

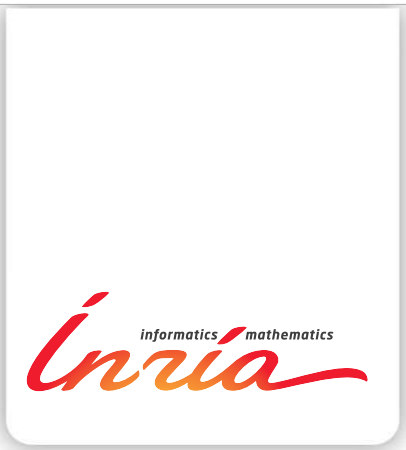
4 A state-based semantics for CCSL operators **11**
4.1 CCSL clocks and relations 11
4.2 CCSL bounded expressions 12
4.3 Unbounded relations 15
4.4 Unbounded expressions 16

5 Boundness issues on CCSL specifications **17**

6 Example: CCSL for capturing the architecture, application and allocation **19**
6.1 Application 19
6.2 Execution platform and allocation 21

7 Related work **22**

8 Conclusion **24**



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399